



PARAMETER TOOLS

Supplement to Patch Description

Patch XT*7.3*26

August 2001

Department of Veterans Affairs
VHA OI System Design & Development (SD&D)
Information Infrastructure Service (IIS)

Preface

This supplemental documentation is intended for use in conjunction with the Parameter Tools patch (XT*7.3*26). This documentation explains the functions available with the use of the Parameter Tools and describes the APIs that are part of the patch. It combines information from the patch description and two Database Integration Agreements (DBIAs) (2263 and 2236) as well as providing additional explanatory material and a generic example to illustrate the use of the Parameter Tools.

In brief, the Parameter Tools patch provides a method of managing the definition, assignment, and retrieval of parameters for **VISTA** packages.

The intended audience for this documentation is Veterans Health Information Systems and Technology Architecture (**VISTA**) sites' Information Resource Management (IRM), VHA Office of Information (OI) System Design & Development (SD&D), and National **VISTA** Support (NVS).

Table of Contents

Preface.....	iii
Introduction.....	1
Background	3
Description.....	5
Entity	5
Parameter	6
Instance	6
Value	6
Parameter Template	7
Why Would You Use Parameter Tools?	9
Example	11
Supported Parameter Tool Entry Points: ^XPAR	13
EN^XPAR: (Entity,Parameter,Instance,Value,,Error)	13
ADD^XPAR: (Entity Parameter,Instance,Value,,Error)	15
CHG^XPAR: (Entity,Parameter,Instance,Value,,Error).....	17
DEL^XPAR: (Entity,Parameter,Instance,,Error).....	19
NDEL^XPAR: (Entity,Parameter,,Error)	21
REP^XPAR: (Entity,Parameter,CurrentInstance,NewInstance,,Error)	23
\$\$GET^XPAR: (Entity,Parameter,Instance,Format).....	25
GETLST^XPAR: (.List,Entity,Parameter,Format,,Error)	26
GETWP^XPAR: (ReturnedText,Entity,Parameter,Instance,Error)	28
PUT^XPAR: (Entity,Parameter,Instance,Value,Error).....	30
ENVAL^XPAR: (List,Parameter,Instance,,Error).....	31
Supported Parameter Tool Entry Points: ^XPAREDIT	33
EN^XPAREDIT.....	33
TED^XPAREDIT(Template,ReviewFlags,AllEntities)	34
TEDH^XPAREDIT(Template,ReviewFlag,AllEntities)	35
EDITPAR^XPAREDIT(Parameter)	36
GETPAR^XPAREDIT(.Variable)	37
GETENT^XPAREDIT(.Entity,Parameter,,OnlyOne?)	38
EDIT^XPAREDIT(Entity,Parameter)	39
BLDLST^XPAREDIT(.List,Parameter).....	40
INDEX	41

Table of Contents

Introduction

VISTA packages were designed to be used in a variety of ways. Many aspects of hospital activity vary from one hospital to another and thus there are many possible ways a package may be used which also vary from one institution to another. Each site has its own requirements – its own settings for each package. IRM staff must modify the package parameters to fit their requirements.

Previously each package had its own files and options but no two packages had the site parameters set up the same way or found in the same place. Thus when a new package was released, each site would have to look for the location where the settings were stored for that package. Next they would have to look to see what settings were available and how to set them. Very little about the parameters was uniform from package to package.

With the Computerized Patient Record System (CPRS) package, the idea was born that a parameter file could be created to export with the package. The CPRS parameter file and parameter utility were subsequently modified to create a generic method of exporting and installing other **VISTA** packages. Most developers were willing to abandon previous methods and use this tool for packages they were developing.

Parameter Tools was designed as a method of managing the definition, assignment, and retrieval of parameters for **VISTA** packages. A parameter may be defined for various levels at which you want to allow the parameter described (e.g., package level, system level, division level, location level, user level).

Introduction

Background

Whenever you have an entity with many attributes that apply to it, you can either (1) make one big relation to represent that entity, or (2) create a "binary" relation to represent the entity. In the latter case, the relation consists of two columns (thus the term binary), one representing the attribute and the other representing the value for that attribute. So each tuple of the relation represents a single attribute and its associated value. Note that this works only when the individual attributes are independent observations (have no dependencies on anything other than the key that identifies the entity). Such a relation tends to look a lot like a Windows INI file.

Most of the **VISTA** parameter files were very long lists of independent values that pertained to a single entity. In most cases, this entity was the site or system on which the software was running [similar to an INI file]. In other cases, however, the parameter files had multiples that made things more complex. These multiples generally allow parameters to be defined at levels more specific than the site (by divisions or hospital location, for example). It seems best to accommodate this by using both an entity identifier and parameter together to name any given value. This yields a relation with a compound key:

Entity | Parameter = Value

Finally, it seems that multiple-valued parameters (collection times, for example) occur often enough that it is worthwhile to add a field to identify the parameter instance. So the relation becomes:

Entity | Parameter | Instance = Value

This is the relation that the PARAMETERS file (8989.5) is intended to represent.

Package parameter files frequently maintain parameters that apply to the site, a division, or a location. In addition, many parameters that apply to individual users are kept in the NEW PERSON file. Also, many parameter values are hard-coded in individual package routines for the case when the site has not set up a value for a given parameter. Entity, then, is implemented as a variable pointer.

A given parameter may occur for a variety of entities. In fact, we frequently need to obtain the value of a parameter by following an entity 'chain'. For example, the 'Add Orders' menu a CPRS user sees may be defined at various levels. Initially, a site generally creates a custom 'Add Orders' menu. Later, hospital locations may each build a custom menu that more specifically meets their needs. Individual users may also have their own 'Add Orders' menus. If no site configuration has been done, the 'Add Orders' menu exported with OE/RR is used. So, when OE/RR needs to display an 'Add Orders' menu, a chain is followed that looks first to see if the user has their own menu. Next, the current location is checked, followed by the site. Finally, if no values exist, the package default menu is used.

In the PARAMETER DEFINITION file (#8989.51), a multiple lists which entities are valid with a given parameter. These entities are also assigned a precedence, so that it is possible to write functions that will 'chain' through entities until a value is found, using the proper sequence.

Background

Description

This patch contains a new developer toolset which allows creation of package parameters in a central location. Database Integration Agreements (DBIAs) 2263 and 2336 define the supported entry points for this application. Kernel patch XU*8*201 will allow KIDS to transport the parameters.

Parameter Tools is a generic method of handling parameter definition, assignment, and retrieval. A parameter may be defined for various entities where an entity is the level at which you want to allow the parameter defined (e.g., package level, system level, division level, location level, user level, etc.). A developer may then determine in which order the values assigned to given entities are interpreted. Following are some basic definitions used by Parameter Tools.

Entity

An entity is a level at which you can define a parameter. The entities allowed are stored in the Parameter Entity file (#8989.518). Entries in this file will be maintained by Toolkit patches.

The list of allowable entries is as follows:

PREFIX	MESSAGE	POINTS TO FILE
PKG	Package	Package (9.4)
SYS	System	Domain (4.2)
DIV	Division	Institution (4)
SRV	Service	Service/Section (49)
LOC	Location	Hospital Location (44)
TEA	Team	Team (404.51)
CLS	Class	Usr Class (8930)
USR	User	New Person (200)
BED	Room-Bed	Room-Bed (405.4)
OTL	Team (OE/RR)	OE/RR List (101.21)

Package, as an entity, allows the package defaults to be handled the same way as other parameters rather than reside in hard code.

System, Division, Location, and User are frequent entries in existing package parameter files (or additions to the New Person file).

Description

Service, Team, and Class are referenced frequently by parameters that pertain to Notifications.

The process of exporting a package using this kind of parameters file involves sending:

1. the parameter definitions that belong to the package (entries in the PARAMETER DEFINITION file)
2. actual parameter instances that point to the package (entries in the PARAMETERS file that have an entity that matches the package)

All the other entries in the PARAMETERS file (those that correspond to entities other than package) would never be exported, as they are only valid for the system on which they reside.

Parameter

A parameter is the actual name under which values are stored. The name of the parameter must be namespaced and it must be unique and start with two uppercase characters. Parameters can be defined to store the typical package parameter data (e.g., the default add order screen in OE/RR), but they can also be used to store GUI application screen settings a user has selected (e.g., font or window width). With each parameter, a more readable display name can also be defined. When a parameter is defined, the entities which may set that parameter are also defined. The definition of parameters is stored in the PARAMETER DEFINITION file (#8989.51).

Instance

An instance is a unique value assigned to an entity/parameter combination. For most parameters, there will only be one instance, that is, instance does not apply and is simply set to '1'.

However, a parameter may be multi-valued - it may have more than one instance. There can be more than one value assigned to the parameter as it relates to a specific entity. An example of this would be lab collection times at a division. For a single entity (division in this case), multiple collection times may exist. Each collection time would be assigned a unique instance.

A parameter is not considered multi-valued if it may apply to several entities, but for each entity only one value of the parameter exists. For example, 'maximum days for a lab order' may be set for every location in the hospital. But since there is only one value for each location, 'maximum days for a lab order' is not multi-valued. When a parameter that is multi-valued is defined, the instance may be defined as numeric, a date/time, a pointer, a set of codes, free text, or yes/no. The validating logic for an instance is defined the same way as for a value.

Value

A value may be assigned to every parameter for the entities allowed in the parameter definition. Values are stored in the PARAMETERS file (#8989.5). Fields in the PARAMETERS file map to DIR fields. DIR is used to validate the data. Values may be numeric, a date/time, a pointer, a set of codes, free text, yes/no, or word processing type.

Parameter Template

A parameter template is similar to an input template. It contains a list of parameters that can be entered through an input session (e.g., an option). Templates are stored in the Parameter Template file (#8989.52). Entries in this file must also be namespaced.

There are two input templates for adding parameter definitions:

XPAR SINGLE VALUED CREATE	for adding/editing parameters that will be single valued
XPAR MULTI VALUED CREATE	for adding/editing parameters that will be multiple valued

Description

Why Would You Use Parameter Tools?

The reason a developer would use parameter tools is to allow a hierarchical designation of a parameter value. So, rather than many parameters that exist now which are just for the system level or just for a particular clinic, parameter tools allows you to define (1) different levels at which the parameter can be set and (2) in what priority the values are used.

Take for example setting up a default order menu for a person. Each facility may have a default order menu for their primary care clinicians. Each division may have one that is slightly different if their practices vary enough. For each location, they may set up a different order menu so that users working in a cardiology clinic get a different set of possible orders than those in a dermatology clinic. And there may be reasons to give one specific person a different order menu because they are authorized to prescribe additional medications, because they tend to practice in a different flow, or for other reasons. It's one parameter, but it allows the parameter to be set for multiple entities (at multiple levels). Those entities are defined in the DBIA, but can include package (which only developers should set - these are default export values), system (whole medical facility), division, location, room-bed, team, provider, etc.

The PARAMETER DEFINITION file defines what entities are allowed to be used for a parameter and in which order they are resolved (individual takes precedence over location takes precedence over division takes precedence over system which takes precedence over package). Sometimes you would want to create defaults for your medical center, but allow users in a certain area to customize what they see and do for their particular role.

XPAR finds the appropriate value based on the parameter definitions and settings that may exist. This way, the developer does not need to look at multiple different location or person files to determine how the software should operate.

With integrations, this is even more important because it allows facilities to integrate but, at the same time, continue some business practices based on parameters set at the division level rather than at the system level.

Why Would You Use Parameter Tools?

Example

The following is a simple example of a way you might use the Parameter Tools.

Suppose you needed a parameter that could be set as a default for the system (account) and also overridden for a given user. The old way was to add a field to a package site file (for example, the Kernel SYSTEM PARAMETERS) and then add a similar field to the NEW PERSON file. This situation is a perfect use of the Parameter Tools.

1. You need the equivalent to a DD entry. This goes into the PARAMETER DEFINITION file (#8989.51). In this case we need a yes/no set of codes. So this is what you set up:

```
Name: XUS-XUP VPE
DISPLAY TEXT: Drop into VPE
MULTIPLE VALUED: No
VALUE DATA TYPE: yes/no
VALUE HELP: Should XUP drop the user into the VPE environment?
Description...
PRECEDENCE: 1      ENTITY FILE: USER
PRECEDENCE: 2      ENTITY FILE: SYSTEM
```

This last item gives the order that values are looked for and/or returned. You want a USER value (File 200) if there is one; otherwise a SYSTEM value (File 4.2). It also gives the entities that are allowed to have values of this data. In the place of SYSTEM, you could have used PACKAGE.

2. Now you can use ^XPAREDIT to enter a value for your new parameter.

D ^XPAREDIT

--- Edit Parameter Values ---

Select PARAMETER DEFINITION NAME: XUS-XUP VPE Drop into VPE

XUS-XUP VPE may be set for the following:

1	User	USR	[choose from NEW PERSON]
2	System	SYS	[NXT.KERNEL.ISC-SF.VA.GOV]

Enter selection: 2 System NXT.KERNEL.ISC-SF.VA.GOV

----- Setting XUS-XUP VPE for System: NXT.KERNEL.ISC-SF.VA.GOV -----

Value: NO

...

3. Now how do you get this value out in your program?

```
S X=$$GET^XPAR("USR^SYS","XUS-XUP VPE",1,"Q") ;X will be null, 0 or 1.
```

For the first parameter, you want a value from USR (user / New Person) or SYS (system)
Next, this is the name of the parameter: "XUS-XUP VPE"

Next, in this example you only allow one instance (optional, Defaults to 1 if not passed in).

Last, the format to return: you use "Q" to get, in the quickest manner, the internal value.

Examples

Adding the parameter template with FileMan:

```
Select PARAMETER DEFINITION NAME: XUS-XUP VPE           Drop into VPE

NAME: XUS-XUP VPE//                                     DISPLAY TEXT: Drop into VPE//
MULTIPLE VALUED: No//                                 INSTANCE TERM:
VALUE TERM:
PROHIBIT EDITING:
VALUE DATA TYPE: yes/no//                           VALUE DOMAIN:
VALUE HELP: Should XUP drop the user into the VPE environment.
VALUE VALIDATION CODE:
VALUE SCREEN CODE:
INSTANCE DATA TYPE:
INSTANCE DOMAIN:
INSTANCE HELP:
INSTANCE VALIDATION CODE:
INSTANCE SCREEN CODE:
DESCRIPTION:
  1>This parameter controls if a user when exiting XUP is dropped into
  2> VPE or right to the ">" prompt.
EDIT Option:
Select PRECEDENCE: 2//                               PRECEDENCE: 2//          ENTITY FILE: SYSTEM//
Select PRECEDENCE:
```

Supported Parameter Tool Entry Points: ^XPAR

Following is a description of the callable entry points in routine XPAR.

EN^XPAR: (Entity,Parameter,Instance,Value,,Error)

This entry point will:

1. add the value as a new entry to the Parameters file if the Entity|Parameter|Instance combination does not already exist,
2. change the value assigned to the parameter if the Entity|Parameter|Instance combination already exists, or
3. delete the parameter instance if the value assigned is "@".

Input Variables

- | | |
|---------------|--|
| Entity | (REQUIRED) The entity may be set to:
1) the internal variable pointer (nnn;GLO(123,)
2) the external format of the variable pointer using the 3 character prefix
(prefix.entryname), or
3) the prefix alone to set the parameter based on current entity selected. |
|---------------|--|

This will work for the following entities:

- "USR" - uses current value of DUZ
- "DIV" - uses current value of DUZ(2)
- "SYS" - uses system (domain)
- "PKG" - uses the package to which the parameter belongs

Entity: may also be a list of entities delimited by '^' or the word 'ALL'. The list of entities is processed in left to right order and the first existing instance is returned. The word 'ALL' uses the entity precedence defined by the PARAMETER DEFINITION file and looks for values in that order. This is useful when the default values are used.

For example, to search for a parameter instance, using the precedence defined in the PARAMETER DEFINITION file and taking default entity values -

MYENT="ALL"

To do the same as above, but explicitly name a value for one of the entities -

```
MYENT="ALL^LOC . PULMONARY CLINIC"
```

To explicitly name the search order and entity values -

```
MYENT="USR^LOC . PULMONARY CLINIC^SYS^DIV^PKG"
```

or using internal values, variable pointers, etc. -

```
MYENT="USR . `1234^LOC . `57^SYS^34;DIC(4,^PKG"
```

Of course, you may also just pass a single entity in external or internal variable pointer format -

```
MYENT="LOC . PULMONARY CLINIC"
```

Parameter (REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).

Instance (OPTIONAL) (defaults to 1 if not passed in)
May be passed in external or internal format. Internal format requires that the value be preceded by the `'' character.

Value (REQUIRED) May be passed in external or internal format. If using internal format for a pointer type parameter, the value must be preceded with the `'' character.).
If the value is being assigned to a word processing parameter, the text may be passed in the subordinate nodes of Value (e.g. Value(1,0)=Text) and the variable "Value" itself may be defined as a title or description of the text.

Output Variables

Error (OPTIONAL) If used, must be passed in by reference. It will return any error condition which may occur. If no error occurs, the value assigned will be 0 (zero). If an error does occur, it will be in the format: "#^errortext".
The # is the number in the FileMan DIALOG file and the errortext describes the error.

ADD^XPAR: (Entity Parameter,Instance,Value,,Error)

This entry point can be called to add a new parameter value.

Definitions of the input and output variables are the same as for EN^XPAR.

Input Variables

- | | |
|---------------|--------------------------------------|
| Entity | (REQUIRED) The entity may be set to: |
|---------------|--------------------------------------|
- 1) the internal variable pointer (nnn;GLO(123,)
 - 2) the external format of the variable pointer using the 3 character prefix (prefix.entryname), or
 - 3) the prefix alone to set the parameter based on current entity selected.
- This will work for the following entities:
- "USR" - uses current value of DUX
- "DIV" - uses current value of DUX(2)
- "SYS" - uses system (domain)
- "PKG" - uses the package to which the parameter belongs

Entity: may also be a list of entities delimited by '^' or the word 'ALL'. The list of entities is processed in left to right order and the first existing instance is returned. The word 'ALL' uses the entity precedence defined by the PARAMETER DEFINITION file and looks for values in that order. This is useful when the default values are used.

For example, to search for a parameter instance, using the precedence defined in the PARAMETER DEFINITION file and taking default entity values -

MYENT="ALL"

To do the same as above, but explicitly name a value for one of the entities -

MYENT="ALL^LOC . PULMONARY CLINIC"

To explicitly name the search order and entity values -

MYENT="USR^LOC . PULMONARY CLINIC^SYS^DIV^PKG"

or using internal values, variable pointers, etc. -

```
MYENT="USR. `1234^LOC. `57^SYS^34;DIC(4, ^PKG"
```

Of course, you may also just pass a single entity in external or internal variable pointer format -

```
MYENT="LOC. PULMONARY CLINIC"
```

Parameter (REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).

Instance (OPTIONAL) (defaults to 1 if not passed in)

May be passed in external or internal format. Internal format requires that the value be preceded by the ` character.

Value (REQUIRED) May be passed in external or internal format. If using internal format for a pointer type parameter, the value must be preceded with the ` character.).

If the value is being assigned to a word processing parameter, the text may be passed in the subordinate nodes of Value (e.g. Value(1,0)=Text) and the variable "Value" itself may be defined as a title or description of the text.

Output Variables

Error (OPTIONAL) If used, must be passed in by reference. It will return any error condition which may occur. If no error occurs, the value assigned will be 0 (zero). If an error does occur, it will be in the format: "#^errortext".

The # is the number in the FileMan DIALOG file and the errortext describes the error.

CHG^XPAR: (Entity,Parameter,Instance,Value,.Error)

This entry point can be called to change an existing parameter value.

Definitions of the input and output variables are the same as for EN^XPAR.

Input Variables

- | | |
|---------------|--------------------------------------|
| Entity | (REQUIRED) The entity may be set to: |
|---------------|--------------------------------------|
- 1) the internal variable pointer (nnn;GLO(123,)
 - 2) the external format of the variable pointer using the 3 character prefix (prefix.entryname), or
 - 3) the prefix alone to set the parameter based on current entity selected.

This will work for the following entities:

- "USR" - uses current value of DUZ
- "DIV" - uses current value of DUZ(2)
- "SYS" - uses system (domain)
- "PKG" - uses the package to which the parameter belongs

Entity: may also be a list of entities delimited by '^' or the word 'ALL'. The list of entities is processed in left to right order and the first existing instance is returned. The word 'ALL' uses the entity precedence defined by the PARAMETER DEFINITION file and looks for values in that order. This is useful when the default values are used.

For example, to search for a parameter instance, using the precedence defined in the PARAMETER DEFINITION file and taking default entity values -

```
MYENT="ALL"
```

To do the same as above, but explicitly name a value for one of the entities -

```
MYENT="ALL^LOC . PULMONARY CLINIC"
```

To explicitly name the search order and entity values -

```
MYENT="USR^LOC . PULMONARY CLINIC^SYS^DIV^PKG"
```

or using internal values, variable pointers, etc. -

```
MYENT="USR. `1234^LOC. `57^SYS^34;DIC(4,^PKG"
```

Of course, you may also just pass a single entity in external or internal variable pointer format -

```
MYENT="LOC.PULMONARY CLINIC"
```

Parameter (REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).

Instance (OPTIONAL) (defaults to 1 if not passed in)

May be passed in external or internal format. Internal format requires that the value be preceded by the ` character.

Value (REQUIRED) May be passed in external or internal format. If using internal format for a pointer type parameter, the value must be preceded with the ` character.).

If the value is being assigned to a word processing parameter, the text may be passed in the subordinate nodes of Value (e.g. Value(1,0)=Text) and the variable "Value" itself may be defined as a title or description of the text.

Output Variables

Error (OPTIONAL) If used, must be passed in by reference. It will return any error condition which may occur. If no error occurs, the value assigned will be 0 (zero). If an error does occur, it will be in the format: "#^.The # is the number in the FileMan DIALOG file and the errortext describes the error.

DEL^XPAR: (Entity,Parameter,Instance,Error)

This entry point can be called to delete an existing parameter value.

Definitions of the input and output variables are the same as for EN^XPAR.

Input Variables

- | | |
|---------------|--------------------------------------|
| Entity | (REQUIRED) The entity may be set to: |
|---------------|--------------------------------------|
- 1) the internal variable pointer (nnn;GLO(123,)
 - 2) the external format of the variable pointer using the 3 character prefix (prefix.entryname), or
 - 3) the prefix alone to set the parameter based on current entity selected.

This will work for the following entities:

- "USR" - uses current value of DUZ
- "DIV" - uses current value of DUZ(2)
- "SYS" - uses system (domain)
- "PKG" - uses the package to which the parameter belongs

Entity: may also be a list of entities delimited by '^' or the word 'ALL'. The list of entities is processed in left to right order and the first existing instance is returned. The word 'ALL' uses the entity precedence defined by the PARAMETER DEFINITION file and looks for values in that order. This is useful when the default values are used.

For example, to search for a parameter instance, using the precedence defined in the PARAMETER DEFINITION file and taking default entity values -

```
MYENT="ALL"
```

To do the same as above, but explicitly name a value for one of the entities -

```
MYENT="ALL^LOC . PULMONARY CLINIC"
```

To explicitly name the search order and entity values -

```
MYENT="USR^LOC . PULMONARY CLINIC^SYS^DIV^PKG"
```

or using internal values, variable pointers, etc. -

```
MYENT="USR. `1234^LOC. `57^SYS^34;DIC(4,^PKG"
```

Of course, you may also just pass a single entity in external or internal variable pointer format -

```
MYENT="LOC.PULMONARY CLINIC"
```

Parameter (REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).

Instance (OPTIONAL) (defaults to 1 if not passed in)
May be passed in external or internal format. Internal format requires that the value be preceded by the ` character.

Output Variables

Error (OPTIONAL) If used, must be passed in by reference. It will return any error condition which may occur. If no error occurs, the value assigned will be 0 (zero). If an error does occur, it will be in the format: "#^. The # is the number in the FileMan DIALOG file and the errortext describes the error.

NDEL^XPAR: (Entity,Parameter,.Error)

This entry point can be called to delete the value for all instances of a parameter for a given entity.

Definitions of the input and output variables are the same as for EN^XPAR.

Input Variables

- | | |
|---------------|--------------------------------------|
| Entity | (REQUIRED) The entity may be set to: |
|---------------|--------------------------------------|
- 1) the internal variable pointer (nnn;GLO(123,)
 - 2) the external format of the variable pointer using the 3 character prefix (prefix.entryname), or
 - 3) the prefix alone to set the parameter based on current entity selected.

This will work for the following entities:

- "USR" - uses current value of DUX
- "DIV" - uses current value of DUX(2)
- "SYS" - uses system (domain)
- "PKG" - uses the package to which the parameter belongs

Entity: may also be a list of entities delimited by '^' or the word 'ALL'. The list of entities is processed in left to right order and the first existing instance is returned. The word 'ALL' uses the entity precedence defined by the PARAMETER DEFINITION file and looks for values in that order. This is useful when the default values are used.

For example, to search for a parameter instance, using the precedence defined in the PARAMETER DEFINITION file and taking default entity values -

```
MYENT="ALL"
```

To do the same as above, but explicitly name a value for one of the entities -

```
MYENT="ALL^LOC . PULMONARY CLINIC"
```

To explicitly name the search order and entity values -

```
MYENT="USR^LOC . PULMONARY CLINIC^SYS^DIV^PKG"
```

or using internal values, variable pointers, etc. -

Supported Parameter Tool Entry Points: ^XPAR

```
MYENT="USR. `1234^LOC. `57^SYS^34;DIC(4,^PKG"
```

Of course, you may also just pass a single entity in external or internal variable pointer format -

```
MYENT="LOC.PULMONARY CLINIC"
```

Parameter (REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).

Output Variables

Error (OPTIONAL) If used, must be passed in by reference. It will return any error condition which may occur. If no error occurs, the value assigned will be 0 (zero). If an error does occur, it will be in the format: "#^. The # is the number in the FileMan DIALOG file and the errortext describes the error.

REP^XPAR: (Entity,Parameter,CurrentInstance,NewInstance,.Error)

This entry point will allow a developer to replace the value of an instance with another value.

Definitions of the input and output variables are the same as for EN^XPAR.

Input Variables

Entity

(REQUIRED) The entity may be set to:

- 1) the internal variable pointer (nnn;GLO(123,)
- 2) the external format of the variable pointer using the 3 character prefix (prefix.entryname), or
- 3) the prefix alone to set the parameter based on current entity selected.

This will work for the following entities:

- "USR" - uses current value of Duz
- "DIV" - uses current value of Duz(2)
- "SYS" - uses system (domain)
- "PKG" - uses the package to which the parameter belongs

Entity: may also be a list of entities delimited by '^' or the word 'ALL'. The list of entities is processed in left to right order and the first existing instance is returned. The word 'ALL' uses the entity precedence defined by the PARAMETER DEFINITION file and looks for values in that order. This is useful when the default values are used.

For example, to search for a parameter instance, using the precedence defined in the PARAMETER DEFINITION file and taking default entity values -

```
MYENT="ALL"
```

To do the same as above, but explicitly name a value for one of the entities -

```
MYENT="ALL^LOC . PULMONARY CLINIC"
```

To explicitly name the search order and entity values -

```
MYENT="USR^LOC . PULMONARY CLINIC^SYS^DIV^PKG"
```

or using internal values, variable pointers, etc. -

```
MYENT="USR. `1234^LOC. `57^SYS^34;DIC(4,^PKG"
```

Of course, you may also just pass a single entity in external or internal variable pointer format -

```
MYENT="LOC.PULMONARY CLINIC"
```

Parameter (REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).

CurrentInstance (REQUIRED) The instance for which the value is currently defined.

NewInstanc (REQUIRED) The instance for which you want to assign the value currently assigned to CurrentInstance.

Output Variables

Error (OPTIONAL) If used, must be passed in by reference. It will return any error condition which may occur. If no error occurs, the value assigned will be 0 (zero). If an error does occur, it will be in the format: "#^.The # is the number in the FileMan DIALOG file and the errortext describes the error.

\$\$GET^XPAR: (Entity,Parameter,Instance,Format)

This call will allow you to retrieve the value of a parameter. The value is returned from this extrinsic function in the format defined by the variable Format (see below).

Input Variables

Entity	Entity is defined as the single entity or group of entities you want to look at in order to retrieve the value. Entities may be passed in internal or external format (e.g. LOC.PULMONARY or LOC.'57 or 57;SC()). The list of entities in this variable may be defined as follows: <ol style="list-style-type: none"> 1. A single entity to look at (e.g. LOC.PULMONARY). 2. The word "ALL" which will tell the utility to look for values assigned to the parameter using the entity precedence defined in the PARAMETER DEFINITION file. 3. A list of entities you want to search (e.g. "USR^LOC^SYS^PKG"). The list is searched from left to right with the first value found returned. 4. Items 2 or 3 with specific entity values referenced such as: <ul style="list-style-type: none"> • ALL^LOC.PULMONARY - to look at the defined entity precedence, but when looking at location, only look at the PULMONARY location. • USR^LOC.PULMONARY^SYS^PKG - to look for values for all current user, PULMONARY location, system, or package).
Parameter	(REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).
Instance	(OPTIONAL) Defaults to 1 if not passed in. May be passed in external or internal format. Internal format requires that the value be preceded by the ` character).
Format	(OPTIONAL) Defaults to "Q" if not defined. Format determines how the value is returned. It can be set to the following: "Q" - returns the value in the quickest manner - internal format "E" - returns external value "B" - returns internal^external value

GETLST^XPAR: (.List,Entity,Parameter,Format,.Error)

This entry point is similar to \$\$GET^XPAR *except* this will return ALL instances of a parameter.

Input Variables

Entity	Entity is defined as the single entity or group of entities you want to look at in order to retrieve the value. Entities may be passed in internal or external format (e.g. LOC.PULMONARY or LOC.'57 or 57;SC()). The list of entities in this variable may be defined as follows: <ol style="list-style-type: none">5. A single entity to look at (e.g. LOC.PULMONARY).6. The word "ALL" which will tell the utility to look for values assigned to the parameter using the entity precedence defined in the PARAMETER DEFINITION file.7. A list of entities you want to search (e.g. "USR^LOC^SYS^PKG"). The list is searched from left to right with the first value found returned.8. Items 2 or 3 with specific entity values referenced such as:<ul style="list-style-type: none">• ALL^LOC.PULMONARY - to look at the defined entity precedence, but when looking at location, only look at the PULMONARY location.• USR^LOC.PULMONARY^SYS^PKG - to look for values for all current user, PULMONARY location, system, or package).
Parameter	(REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).
Instance	(OPTIONAL) Defaults to 1 if not passed in. May be passed in external or internal format. Internal format requires that the value be preceded by the "" character).
Format	This variable defines how the data is returned from this call. It may be set to any one of the following: <ul style="list-style-type: none">• "Q" for the quickest value: List(#)=internalinstance^internalvalue• "E" for the external value: List(#)=externalinstance^externalvalue• "B" for both internal and external values:<ul style="list-style-type: none">• List(#,"N")=internalvalue^externalinstance• List(#,"V")="internalvalue^externalvalue• "N" for external instance: List(#,"N")=internalvalue^externalinstance

Output Variables

- | | |
|--------------|---|
| List | The array passed as List will be returned with all of the possible values assigned to the parameter. See variable description for Format to see how this data can be returned. |
| Error | Returns 0 if no error was encountered. If an error does occur, it returns an error code in the format: "#^The # is the number in the FileMan DIALOG file and the errortext describes the error. |

GETWP^XPAR: (ReturnedText,Entity,Parameter,Instance,Error)

This call returns word processing text in ReturnedText. ReturnedText itself contains the value field, which is free text that may contain a title, description, or other text. The word processing text is returned in ReturnedText(#,0). (See Example in Returned Text below.)

Input Variables

ReturnedText This variable is defined as the name of an array in which you want the text returned. .ReturnedText will be set to the title, description, etc. The actual word processing text will be returned in ReturnedText(#,0). Example:

```
ReturnedText="Select Notes Help"  
ReturnedText(1,0)="To select a progress note from the list,"  
ReturnedText(2,0)="click on the date/title of the note."
```

Entity Entity is defined as the single entity or group of entities you want to look at in order to retrieve the value. Entities may be passed in internal or external format (e.g. LOC.PULMONARY or LOC.'57 or 57;SC()). The list of entities in this variable may be defined as follows:

a single entity to look at (e.g. LOC.PULMONARY).
the word "ALL" which will tell the utility to look for values assigned to the parameter using the entity precedence defined in the PARAMETER DEFINITION file.

A list of entities you want to search (e.g. "USR^LOC^SYS^PKG"). The list is searched from left to right with the first value found returned.

- 1) Items 2 or 3 with specific entity values referenced such as:
 - ALL^LOC.PULMONARY - to look at the defined entity precedence, but when looking at location, only look at the PULMONARY location.
 - USR^LOC.PULMONARY^SYS^PKG - to look for values for all current user, PULMONARY location, system, or package.

Parameter (REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).

Instance (OPTIONAL) Defaults to 1 if not passed in.

May be passed in external or internal format. Internal format requires that the value be preceded by the "" character).

Output Variables

Error Returns 0 if no error was encountered. If an error does occur, it returns an error code in the format: "#^. The # is the number in the FileMan DIALOG file and the errortext describes the error.

ReturnedText This variable is defined as the name of an array in which you want the text returned. .ReturnedText will be set to the title, description, etc. The actual word processing text will be returned in ReturnedText(#,0).

Example:

```
D GETWP^XPAR(.X,"PKG","ORW HELP","1stNotes",.ERR)
```

might return:

X="Select Notes Help"

X(1,0)="To select a progress notes from"

X(2,0)="the list, click on the date/title"

X(3,0)="of the note."

PUT^XPAR: (Entity,Parameter,Instance,Value,Error)

This entry point can be called to add or update a parameter instance and bypass the input transforms.

Input Variables

Entity	Entity is defined as the single entity or group of entities you want to look at in order to retrieve the value. Entities may be passed in internal or external format (e.g. LOC.PULMONARY or LOC.'57 or 57;SC()). The list of entities in this variable may be defined as follows: a single entity to look at (e.g. LOC.PULMONARY). the word "ALL" which will tell the utility to look for values assigned to the parameter using the entity precedence defined in the PARAMETER DEFINITION file. A list of entities you want to search (e.g. "USR^LOC^SYS^PKG"). The list is searched from left to right with the first value found returned. 1) Items 2 or 3 with specific entity values referenced such as: <ul style="list-style-type: none">• ALL^LOC.PULMONARY - to look at the defined entity precedence, but when looking at location, only look at the PULMONARY location.• USR^LOC.PULMONARY^SYS^PKG - to look for values for all current user, PULMONARY location, system, or package.
Parameter	(REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).
Instance	(OPTIONAL) Defaults to 1 if not passed in. May be passed in external or internal format. Internal format requires that the value be preceded by the `'' character).
Value	(REQUIRED) May be passed in external or internal format. If using internal format for a pointer type parameter, the value must be preceded with the `'' character.). If the value is being assigned to a word processing parameter, the text may be passed in the subordinate nodes of Value (e.g. Value(1,0)=Text) and the variable "Value" itself may be defined as a title or description of the text.

Output Variables

Error	Returns 0 if no error was encountered. If an error does occur, it returns an error code in the format: "#^.The # is the number in the FileMan DIALOG file and the errortext describes the error.
--------------	--

ENVAL^XPAR: (List,Parameter,Instance,.Error)

This entry point will return all parameter instances.

The definition of the input and output variables to this call are the same as for GETLST and \$\$GET.

Input Variables

Parameter	(REQUIRED) Identifies the name or internal entry number of the parameter as defined in the PARAMETER DEFINITION file (#8989.51).
Instance	(OPTIONAL) Defaults to 1 if not passed in. May be passed in external or internal format. Internal format requires that the value be preceded by the ``` character).

Output Variables

List	The array passed as List will be returned with all of the possible values assigned to the parameter. See variable description for Format to see how this data can be returned.
Error	Returns 0 if no error was encountered. If an error does occur, it returns an error code in the format: "#^. The # is the number in the FileMan DIALOG file and the errortext describes the error.

Supported Parameter Tool Entry Points: ^XPAR

Supported Parameter Tool Entry Points: ^XPAREDIT

Following is a list of calls to XPAREDIT. The calls are supported for use with the Parameter Tools and are part of the Parameter Tools component of Toolkit. These calls contain some additional utilities for editing parameters and are covered by DBIA 2236. (See DBIA 2263 for the main XPAR entry points to this module.)

EN^XPAREDIT

This entry point can be called to prompt the user for a parameter to edit. This is provided as a tool for developers and *is not intended for exported calls* as it allows editing of ANY parameter.

TED^XPAREDIT(Template,ReviewFlags,AllEntities)

This entry point allows editing of parameters defined in a template. The parameters in the template are prompted in FileMan style - prompt by prompt. No dashed line dividers are displayed between each parameter.

Since the dashed line headers are suppressed, it is important to define the VALUE TERM for each parameter in the template, as this is what is used to prompt for the value.

Input Variables

Template	(REQUIRED) The IEN or NAME of an entry in the Parameter Template file (#8989.52).
ReviewFlag	(OPTIONAL) There are two flags (A and B) that can be used individually, together, or not at all. A Indicates that the new values for the parameters in the template are displayed AFTER the prompting is done. B Indicates that the current values of the parameters will be displayed BEFORE editing.
AllEntitie	(OPTIONAL) This is a variable pointer that should be used as the entity for all parameters in the template. If left blank, prompting for the entity is done as defined in the PARAMETER TEMPLATE file.

Output Variables

List	The array passed as List will be returned with all of the possible values assigned to the parameter. See variable description for Format to see how this data can be returned.
Error	Returns 0 if no error was encountered. If an error does occur, it returns an error code in the format: "#^. The # is the number in the FileMan DIALOG file and the errortext describes the error.

TEDH^XPAREDIT(Template,ReviewFlag,AllEntities)

This entry point is similar to the TED^XPAREDIT call except that the dashed line headers ARE shown between each parameter.

It allows editing of parameters defined in a template. The parameters in the template are prompted in FileMan style - prompt by prompt.

Input Variables

Template	(REQUIRED) The IEN or NAME of an entry in the Parameter Template file (#8989.52).
ReviewFlagS	(OPTIONAL) There are two flags (A and B) that can be used individually, together, or not at all. A Indicates that the new values for the parameters in the template are displayed AFTER the prompting is done. B Indicates that the current values of the parameters will be displayed BEFORE editing.
AllEntitie	(OPTIONAL) This is a variable pointer that should be used as the entity for all parameters in the template. If left blank, prompting for the entity is done as defined in the PARAMETER TEMPLATE file.

EDITPAR^XPAREDIT(Parameter)

This entry point can be used to edit a single parameter.

Input Variables

Parameter	Pass as the IEN or the NAME of the entry in the PARAMETER DEFINITION file (#8989.51) which you want to be edited.
------------------	---

GETPAR^XPAREDIT(.Variable)

This entry point will allow the user to select the PARAMETER DEFINITION file entry.

Output Variables

OutputValue Returns the value Y in standard DIC lookup format.

GETENT^XPAREDIT(.Entity,Parameter,OnlyOne?)

This entry point interactively prompts for an entity, based on the definition of a parameter.

Input Variables

Parameter	(REQUIRED) Specifies the parameter for which an entity should be selected. Parameter should contain two pieces: IEN^DisplayNameOfParameter.
------------------	---

Output Variables

Entity	(REQUIRED) Returns the selected entity in variable pointer format.
OnlyOne?	(OPTIONAL) Returns "1" if there is only one possible entity for the value. For example, if the parameter can only be set for the system, OnlyOne?=1. If the parameter could be set for any location, OnlyOne?=0.

EDIT^XPAREDIT(Entity,Parameter)

This entry point interactively edits the instance (if multiple instances are allowed) and the value for a parameter associated with a given entity

Input Variables

Entity	(REQUIRED) Identifies the specific entity for which a parameter may be edited. Entity must be in variable pointer format.
Parameter	(REQUIRED) Identifies the parameter that should be edited. Parameter should contain two pieces: IEN^DisplayNameOfParameter.

Output Variables

List	The array passed as List will be returned with all of the possible values assigned to the parameter. See variable description for Format to see how this data can be returned.
Error	Returns 0 if no error was encountered. If an error does occur, it returns an error code in the format: "#^. The # is the number in the FileMan DIALOG file and the errortext describes the error.

BLDLST^XPAREDIT(.List,Parameter)

This entry point will return, in the array List, all entities allowed for the input Parameter.

Input Variables

List Name of array to receive output.

Parameter IEN of entry in the PARAMETER DEFINITION file.

INDEX

\$\$GET^XPAR	25	TED^XPAREDIT	34
ADD^XPAR	15	TEDH^XPAREDIT	35
Background.....	3	XPAR	13
BLDLST^XPAREDIT	40	\$\$GET	25
CHG^XPAR.....	17	ADD.....	15
DEL^XPAR	19	CHG.....	17
Description.....	5	DEL	19
EDIT^XPAREDIT	39	EN	13
EDITPAR^XPAREDIT	36	ENVAL.....	31
EN^XPAR.....	13	GETLST	26
EN^XPAREDIT.....	33	GETWP	28
ENVAL^XPAR	31	NDEL.....	21
GETENT^XPAREDIT	38	PUT.....	30
GETLST^XPAR	26	REP	23
GETPAR^XPAREDIT	37	XPAREDIT	33
GETWP^XPAR	28	BLDLST	40
Introduction.....	1	EDIT	39
NDEL^XPAR	21	EDITPAR	36
Preface	iii	EN	33
PUT^XPAR	30	GETENT	38
REP^XPAR.....	23	GETPAR	37
Routine		TED	34
XPAR.....	13	TEDH.....	35
XPAREDIT.....	33		

